REMARKS

In the Official Action mailed on **01 October 2007**, the Examiner reviewed claims 1-20. Claims 1-7, 9-16 and 18-20 were rejected under 35 U.S.C. § 102(e) as being anticipated by Rajwar et al. (USPN 7,120,762, hereinafter "Rajwar"). Claims 8 and 17 were rejected under 35 U.S.C. § 103(a) as being unpatentable over Rajwar, in view of Hecht et al. (U.S. Pub. No. 2003/0064808, hereinafter "Hecht").

In the Advisory Action mailed on **22 January 2008**, Examiner provided general definitions of the terms "committing" and "atomically." Examiner then pointed out that Rajwar "describes how locks are predominantly used to commit changes in the background."

In the instant response, Applicant has included the argument made in the response to the Office Action mailed 01 October 2007. Applicant has also responded to Examiner's points from the Advisory Action mailed 22 January 2008. The response to the Office Action mailed 01 October 2007 appears in the first section, while the response to the Advisory Action mailed 22 January 2008 appears in the second section.

Rejections under 35 U.S.C. § 102(e)

Examiner rejected claims 1-20 under 35 U.S.C. § 102(e), asserting that these claims are anticipated by Rajwar. More specifically, Examiner argues that the Rajwar discloses "atomically committing changes made during transactional execution" (see Office Action page 3, sec. 6). Applicant respectfully disagrees because Rajwar discloses only freely buffering results into an L1 cache and then updating the L2 cache with the results in the L1 cache, whereas atomically committing changes made during transactional execution involves (among other operations, as described below) protecting cache lines modified during

transactional execution from interfering accesses while the results from transactional execution are written from a store buffer to the cache lines.

The Rajwar system monitors code execution to detect the beginning of a critical section. When critical section is executed, the Rajwar system elides the lock-acquiring and lock-releasing steps and speculatively executes the critical section (see Rajwar, claim 1 and col. 2, lines 30-33). At the end of the speculative-execution, if there has been no "interruption" to the access of the portion of a common memory used by the system during the speculative execution, the system "commits the speculative execution of the critical section" (see Rajwar, claim 1 and col. 9, lines 46-49). As described in Rajwar, committing the speculative execution involves only "**updating cache L2 with the L1 cache**." More specifically, Rajwar explicitly discloses using the L1 cache as a "buffer" for the speculative execution of the critical section in order to prevent the effects of the critical section from being observed by other processor units and **writing the speculative results from the L1 cache to the L2 cache at the end of the speculative execution** (see Rajwar, col. 8, lines 35-40). Rajwar discloses only "*standard cache protocol messages*" being used to detect conflicts (see Rajwar, col. 8, lines 48-50). Nothing in Rajwar discloses atomically committing changes made during the transactional execution.

In contrast, in embodiments of the present invention, **the system atomically commits the results from the transactional execution**. More specifically, the system starts by **treating cache lines as though they are locked** (see instant application, par. [0076]-[0079]). This means other processes that need to access a cache line must wait until the line is no longer locked before they can access the cache line. Next, the system clears marks from the cache lines in the L1 data cache. The system then commits entries from the store buffer for stores that were generated during transactional execution. As each entry is committed, the corresponding cache line is unlocked. (The system also commits

register file changes using a flash copy.) Atomically committing the results of the transaction **at the end of the transaction** permits processes to share all levels of the cache hierarchy during a transaction (i.e., processes can share the L1).

In addition, the sections cited by Examiner in support of the rejection using Rajwar do not disclose the indicated subject matter. Specifically:

- col. 3, lines 15-17: describe continuing non-transactional execution at the conclusion of speculative execution;
- col. 5, lines 57-60: generally describe atomic instructions (with no discussion of speculative execution that involves using atomic instructions to complete speculative execution); and
- col. 9, lines 45-50: describe updating the L2 cache from the L1 cache.

Applicant respectfully avers that none of these sections (or any other section of Rajwar) are proper support for the rejection under 35 U.S.C. § 102.

In summary, the Rajwar system depends on being able to freely write the results from transactional execution into the L1 cache (or else return to a conventional technique of getting locks for the data structures) and then write those results to the L2 cache at the end of the speculative execution. Nothing in Rajwar discloses atomically committing changes made during the transactional execution.

Response to Argument in Advisory Action

In the Advisory Action mailed on **22 January 2008**, Examiner provided general definitions of the terms "committing" and "atomically." Examiner then pointed out that Rajwar "describes how locks are predominantly used to commit changes in the background. More specifically, Examiner argued that:

> The invention of Rajwar makes better usage of … locks by speculatively executing the instructions and then when the end is reached and no conflict has occurred; the execution is then to be committed permanently. Since atomically is defined as ensuring an instruction cannot be interrupted and

> Rajwar makes a determination whether a conflict has occurred before committing, Rajwar **inherently teaches atomically committing since only non-conflict speculative execution is committed**. Therefore Rajwar adequately teaches the language of the claim (see Advisory Action, continuation sheet, emphasis added).

Applicant respectfully disagrees. Rajwar does not disclose the atomic commission of changes made during transactional execution that is performed in embodiments of the present invention. Hence, because Rajwar does not disclose each and every element of the claimed invention (i.e., Rajwar nowhere discloses "the identical invention … in as complete detail as is contained in the ... claim."), Rajwar cannot anticipate embodiments of the present invention (see MPEP § 2131).

Applicant respectfully points out that the concepts of "atomic" and "committing" are high-level concepts that are well known in computer science. For example, the online encyclopedia Wikipedia includes a reference that describes atomic operations (http://en.wikipedia.org/wiki/Atomic_operation) as well as atomic transactions (http://en.wikipedia.org/wiki/Atomic_transaction). Applicant therefore avers that Rajwar's disclosure of one particular technique for implementing atomic commitment is insufficient for covering each and every possible technique for implementing atomic commitment.

More specifically, Rajwar is expressly directed at a system that **removes lock instructions** present in program code in the interest of performance gains (see Rajwar, col. 2, lines 20-22 and Abstract). Rajwar explicitly discloses **eliding the lock retention instructions** (i.e., **not obtaining locks**) when executing a critical section (see Rajwar, col. 2, lines 20-22, as "*the steps of acquiring and releasing the lock instructions are unnecessary and can be elided*"). The Rajwar system then speculatively executes the instructions in the critical section, writing results into an L1 cache as during normal execution **without locking the cache** (unless speculation has failed, in which case the L1 cache can be locked and the

critical section executed non-speculatively). Rajwar describes using the L1 cache in detail:

> (1) "doing" stores by the critical section to the L1 cache, which serves as a buffer for the speculative execution of the critical section, and **prevents the effects of the instructions of the critical section from being observed by other processor units** (see Rajwar, col. 8, lines 34-39, emphasis added);
>
> (2) squashing the speculative execution (as indicated by process block 70) by flushing the L1 cache 18 (see Rajwar, col. 8, lines 57-60); and
>
> (3) committing the speculative execution "by updating cache L2 with the L1 cache L1" (see Rajwar, col. 9, lines 47-49).

Rajwar does not disclose using load-marking or store-marking to retain speculative results. Instead, Rajwar teaches buffering results in the L1 cache and releasing (or flushing) these results as a group. Moreover, Rajwar nowhere discloses using anything other than ordinary cache protocols to handle the results of speculative execution.

In contrast, when committing the results of a transaction, embodiments of the present invention treat store-marked cache lines as locked, thereby causing other processes to wait to access the store-marked cache lines (see instant application, original claim 3). These embodiments then commit store buffer entries generated during transactional execution to memory. Committing each store buffer entry involves removing the store-mark from, and thereby unlocking, a corresponding store-marked cache line. These embodiments next clear load-marks from cache lines.

Note that the use of load-marks and store-marks enables **one or more other threads and/or processing units to use a cache** while a transactional thread writes the transactional results in the cache. Nowhere does Rajwar describe two processing units writing speculative results into the same L1 cache.

In fact, Rajwar discloses a system that *expressly* relies on **preventing the effects of instructions in a critical section from being observed by other processor units**. In other words, the Rajwar system as disclosed is *inoperable* where more than one thread is simultaneously using the L1 cache.

Accordingly, Applicant has amended independent claims 1, 9, and 19 to clarify that embodiments of the present invention: (1) treat store-marked cache lines as locked, thereby causing other processes to wait to access the store-marked cache lines; (2) commit store buffer entries generated during transactional execution to memory, wherein committing each store buffer entry involves removing the store-mark from, and thereby unlocking, a corresponding store-marked cache line; and (3) clear load-marks from cache lines when atomically committing the results of transactional execution. Support for these amendments can be found in original claim 3 of the instant application. Applicant has also cancelled claims 3 and 12. No new matter has been added.

Hence, Applicant respectfully submits that the independent claims are in condition for allowance. Applicant also submits that the dependent claims that depend upon these independent claims are in condition for allowance and for reasons of the unique combinations recited in such claims.

CONCLUSION

It is submitted that the application is presently in form for allowance. Such action is respectfully requested.

Respectfully submitted,

By /Anthony Jones/
Anthony Jones
Registration No. 59,521

Date: 19 February 2008

Anthony Jones
Park, Vaughan & Fleming LLP
2820 Fifth Street
Davis, CA 95618-7759
Tel: (530) 759-1666
Fax: (530) 759-1665
Email: tony@parklegal.com